Python Fun(damentals)

Venv Usage & Setup

Alexander Rymdeko-Harvey

Obscurity Labs

```
* Directory Structure
* Config Files
+
```

Python `pipenv` build-system

Pipenv is a tool that aims to bring the best of all packaging worlds (bundler, composer, npm, cargo, yarn, etc.) to the Python world.

- You no longer need to use pip and virtualenv separately. They work together.
- Managing a requirements.txt file can be problematic, so Pipenv uses Pipfile and Pipfile.lock to separate abstract dependency declarations from the last tested combination.
- Hashes are used everywhere, always. Security. Automatically expose security vulnerabilities.
- Strongly encourage the use of the latest versions of dependencies to minimize security risks arising from outdated components.
- Give you insight into your dependency graph (e.g. `$ pipenv graph`).

Using `pipenv`

`pipenv` provides a handy set of tools to work with `virtualenv` locally.

We will start with building our very own `env`:

```
$ pipenv --python 3
⠋created virtual environment CPython3.7.5.final.0-64 in 162ms
creator CPython3Posix(dest=/home/killswitch/.local/share/virtualenvs
  01_python3_tooling_build_systems-YCx-KAYf, clear=False, global=False)
  seeder FromAppData(download=False, pip=latest, setuptools=latest, wheel=latest, via=copy, app_data_dir=/home
  killswitch/.local/share/virtualenv/seed-app-data/v1.0.1)
activators BashActivator,CShellActivator,FishActivator,PowerShellActivator,PythonActivator,XonshActivator

Virtualenv location: /home/killswitch/.local/share/virtualenvs/01_python3_tooling_build_systems-YCx-KAYf
requirements.txt found, instead of Pipfile! Converting…
Warning: Your Pipfile now contains pinned versions, if your requirements.txt did.
We recommend updating your Pipfile to specify the "*" version, instead.
```

We now can simply activate our `pipenv` using the following command:

```
$ pipenv shell
(01_python3_tooling_build_systems-YCx-KAYf) $ python --version
```

Using `pipenv` Cont.

`pipenv` provides you the ability to even run commands directly inside the environment without needing to spawn a shell inside the `env`

```
$ pipenv run python --version
```

Another very handy command to reset your environment by deleting the old `env`:

```
$ pipenv --rm
Removing virtualenv (/home/killswitch/.local/share/virtualenvs/01_python3_tooling_build_systems-YCx-KAYf)…
```

Installing packages with `pipenv`

`pipenv` provides us the ability to install our `requirements.txt`

```
$ pipenv install -r requirements
Creating a Pipfile for this project…
Requirements file provided! Importing into Pipfile…
Pipfile.lock not found, creating…
Locking [dev-packages] dependencies…
Locking [packages] dependencies…
Updated Pipfile.lock (959036)!
Installing dependencies from Pipfile.lock (959036)…
  🐍   ████████████████████████████████████████ 2/2 — 00:00:01
To activate this project's virtualenv, run the following:
 $ pipenv shell
```

We can also pin packages to development so when we go to production we don't bring along extra testing packages.

```
$ pipenv install pytest --dev
```

Lab_4.py

Tasking

Using the new `pipenv` command perform the following:

1. Go into the `02_python3_tooling_build_systems/` folder and install pipenv `pip install pipenv`
2. Install and create our `Pipfile` with `pipenv install -r requirements.txt`
3. Ensure you have a `Pipfile` and `Pipfile.lock`

Testing your work

NOTE: you should see Green `PASS` statements indicating you completed the lab

```
$ pipenv shell
$ python lab_4.py
```